Callie Weber
5/28/2024
Foundations of Python Programming
Assignment 07
https://github.com/cweber08/IntrotoProg-Mod07

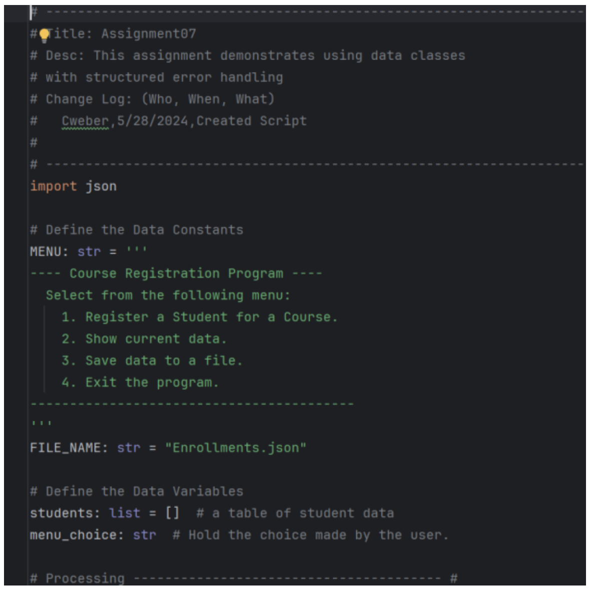
Creating a Python Script

Introduction

This assignment builds on the last by adding common techniques for improving scripts: functions, classes, and the separation of concerns.

Drafting the Code

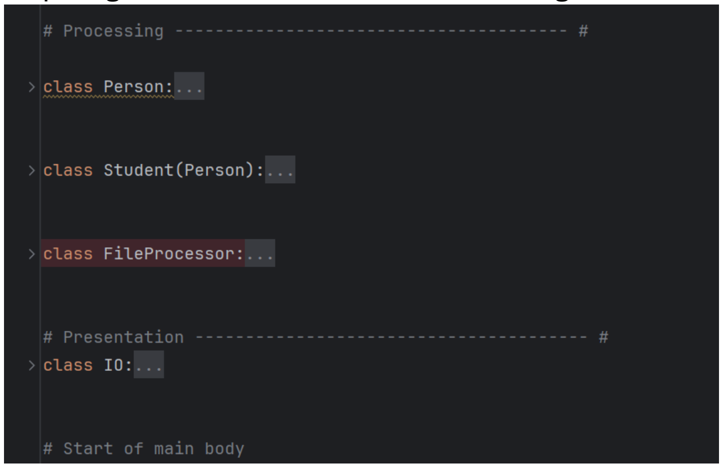
I started this assignment by opening the getting started script provided. It had all the processes laid out and everything pre-defined.



```
# ------------------------------------------------------------------
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   Cweber,5/28/2024,Created Script
#
# ------------------------------------------------------------------
import json

# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.

# Processing --------------------------------------- #
```

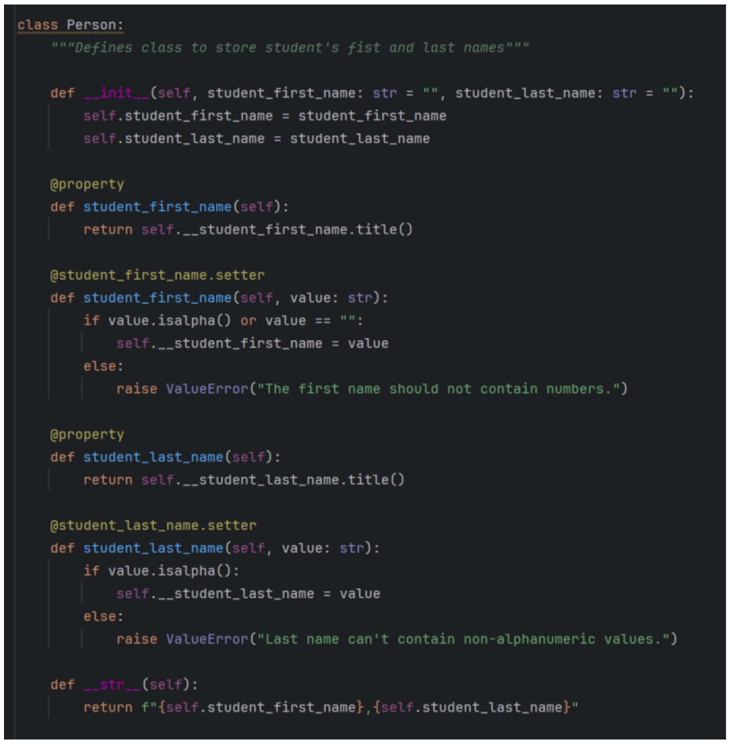
I laid out the 4 classes: Person, Student, File Processor, and IO. The Person class identifies each individual student, the Student class pushes the person data into a data class of all students and their courses. The File Processor reads and writes to the enrollment json, and the IO class obtains the input and outputs of student data:

Callie Weber
5/28/2024
Foundations of Python Programming
Assignment 07
https://github.com/cweber08/IntrotoProg-Mod07



```
# Processing -------------------------------------- #

class Person:...


class Student(Person):...


class FileProcessor:...


# Presentation -------------------------------------- #
class IO:...


# Start of main body
```

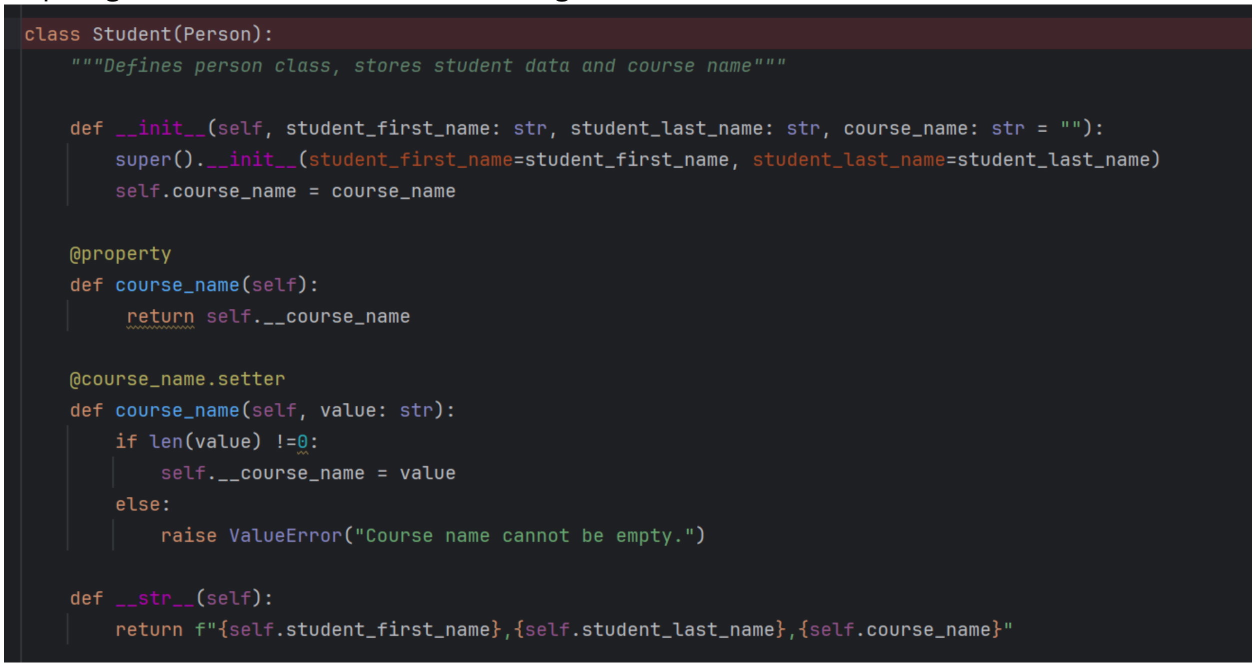
For the Person class we are storing the student's first and last names. The __init__ function is used to assign values to the objects (first/last names). There is also built in error handling for non-alphabetic names:



```python
class Person:
    """Defines class to store student's fist and last names"""

    def __init__(self, student_first_name: str = "", student_last_name: str = ""):
        self.student_first_name = student_first_name
        self.student_last_name = student_last_name

    @property
    def student_first_name(self):
        return self.__student_first_name.title()

    @student_first_name.setter
    def student_first_name(self, value: str):
        if value.isalpha() or value == "":
            self.__student_first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

    @property
    def student_last_name(self):
        return self.__student_last_name.title()

    @student_last_name.setter
    def student_last_name(self, value: str):
        if value.isalpha():
            self.__student_last_name = value
        else:
            raise ValueError("Last name can't contain non-alphanumeric values.")

    def __str__(self):
        return f"{self.student_first_name},{self.student_last_name}"
```

The Student class stores the person data. Again using the __init__ function to assign student names and courses into the student data list:

Callie Weber
5/28/2024
Foundations of Python Programming
Assignment 07
https://github.com/cweber08/IntrotoProg-Mod07



```python
class Student(Person):
    """Defines person class, stores student data and course name"""

    def __init__(self, student_first_name: str, student_last_name: str, course_name: str = ""):
        super().__init__(student_first_name=student_first_name, student_last_name=student_last_name)
        self.course_name = course_name

    @property
    def course_name(self):
         return self.__course_name

    @course_name.setter
    def course_name(self, value: str):
        if len(value) !=0:
            self.__course_name = value
        else:
            raise ValueError("Course name cannot be empty.")

    def __str__(self):
        return f"{self.student_first_name},{self.student_last_name},{self.course_name}"
```

The last two classes were mildly changed. The File Processor class still reads and writes to the json. The IO class appends to a data dictionary of all student data.

<u>Summary</u>

I did not find the demos and labs very helpful this week, now that our code is getting longer, I'm having a very difficult time following along where exactly the demo-er is altering code. Fortunately(?), I'm one of the last submission this week so I could peek at how others manipulated their starter code.